DEVELOPMENT OF A GENERIC VIRUS BEHAVIOURAL DETECTOR: A PREVIEW

J.A. AYENI[+], E.R. ADAGUNODO and A.D. AKINDE
Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria.

(*Submitted*: 22 April 2004; *Accepted*: 31 October 2004)

Abstract

Detecting viruses by observing and monitoring known virus activities while the computer system is in use is known as detection by "behavioural abnormality". In this paper, we examine virus mode of spreading and behaviours, how their infection technique could be used for their detection and present a system for monitoring critical system activities for normal and abnormal behaviours. Generally, viruses spread using either the Operating System or the Computer System as a veritable vehicle to aid the realisation of their motives and detection algorithms are often designed using these spreading modes. The Generic Virus Behavioural Detector (GVBD) is a system (program) that monitors various system activities; reading and writing block of disks and memory and the use of Interrupts. A technique for its realisation is presented.

Keywords: Computer virus, interrupts, handlers, GVBD (Generic Virus Behavioural Detector).

1. Introduction

Viruses are defined as malicious codes that alter, delete, or render files on a given computer system useless (Phillips and Thomas, 1998). They carry out operations that are in no way related to the current requirement of a given computer system. The present day viruses are products of above-average programming skill coupled with some in-depth knowledge of anti-virus techniques (Vprotect, 2002). Traditional anti-virus scanners are not protective enough as an effective combat tool against viruses. The generic virus behavioural detector uses the goal of a virus for its identification and thereby blocking it (the virus) before being triggered.

Common Virus Behaviours
Some known virus behaviours are (Whittington and Wilheims, 2002):

- Interrupt redirection
- Modification of system resources or files (*.exe, *.com and *.sys)
- Unauthorised modification of user files
- Staying resident illegally.

All of these actions are behavioural, and as such, it is possible to define these behaviours in such a way that a program can look for them for possible identification. However, a brief discussion of the spreading mode of viruses is an essential part of this work.

2. Spreading Modes of Viruses

As it is known in practice, a virus can be attached to various program areas. The forms of attachment are referred to as their spreading modes. Viruses do have different spreading modes (Phillip, 1998; Munro, 2002 and Vprotect, 2002) and could be dependent or independent of the machine and/or the operating system.

(a) Machine Depen ent
A spreading mode of a virus can be machine- specific when the program areas which can be infected by the virus depend on the machine. An example is the **boot virus** that depends on the services of the BIOS or the disk controller for its propagation.

(b) Machine Independent
On the other hand, a spreading mode of a virus can be machine independent when the program areas which can be infected are machine independent. An example is a text file, as they could be infected under different machines using the same spreading mode.

(c) Operating System Dependent
A spreading mode of a virus can be operating system dependent when the program areas, which can be infected by the virus, depend on the operating system. An example is the .exe files in DOS.

(d) Operating System Independent
A virus spreading mode is operating system independent if the program areas which can be infected by the virus are independent of the operating

+ *corresponding author* (email: jayeni@oauife.edu.ng)

system. An example is the boot sector virus that uses the BIOS services and not DOS services. However, virus classification is based on these spreading modes and not really on the type of infection. A virus spreading mode often assists in the design of its detection or its detection algorithm.

3. The Behavioural Abnormality Technique

Our approach shall be to observe a given normal system behaviour as opposed to an abnormal behaviour. Of course, normal system behaviour consist of making the system files "***read-only***" with no need for modification except when the system administrator deem it necessary. These modifications could be the upgrading of the Operating System along with its related files, drivers etc. Any program that attempts to do such on its own should trigger a warning, alert the user and be prevented from making such modification. The same applied to Boot Sector writers and the modifications of interrupt vectors. For viruses to remain undetected, they will always attempt to modify their codes or change their locations in memory or the disk. This will be interpreted as a "**behavioural abnormality**". In this paper, we shall deal with four types of "Virus behaviours".

These are:

- i Boot Sector viruses
- ii Interrupt Routine Re-Routers
- iii Viruses that attempt to go "*Resident*" before infection
- iv File Viruses.

(a) The Structure of Viruses
We present a simple structure of a virus (C Language - Fig. 1). The virus is an example of the structure of a file virus (see section 3(e) on File Viruses). The file is often infected at known location within the binary file. The infection is referred to as the virus' signature and often used to eliminate re-infecting an already infected file. That is, if the file (executable) had already been infected, the virus stops action (Fig. 1). We shall briefly analyse the other types of viruses and their methods of infection.

(b) Boot Sector Viruses
This is the type of virus that corrupts the Boot Sector of the disk by the modification technique (Gordon, 1995 and Munro, 2002). The virus infects the system Boot Sector by relocating the original Boot Sector of a disk and copying itself appropriately. This class of virus attempts to redirect the INT 13H (Rom BIOS disk services) and INT 21H (DOS services) to its (virus) codes. Therefore, a first approach to the detection of this class of virus will be to monitor request to modify the disk boot sector.

(c) Interrupt Routine (Vectors) Interceptors
This class of virus redirects the Interrupt Vector associated codes to their own codes. The effect is abnormal system behaviour. Such effects are the keyboard not responding, pointing device pointing abnormally, system resources exhaustion, and also system malfunctioning (Patrick, 1992). We shall approach this abnormal behaviour by monitoring the system's associated Interrupt Vectors and disallowing re-direction.

(d) Memory Resident Viruses
The first task of any given virus on execution is to go resident in memory. This is an essential feature of viruses and hence their capability to perform their malicious tasks. Viruses go resident and await system requests issued by applications to be triggered. Allowing applications to reside in the memory of a computer system is a useful provision by the operating system to enable adequate utilization of the system resources and also service application requests. Most viruses abuse this provision by exploiting its usefulness to achieve their malicious goals. In this regard, we shall be prudent as it is practically unwise to block all programs trying to go resident because most system's utilities reside in memory. It is equally impossible to establish the usefulness, relevance or otherwise of a program going resident without allowing it to execute first. However, a 'launch-code' into memory from another file or a redirection from an EXE or COM file to another section of memory may look suspicious. We shall approach the 'resident virus' problem by examining where the calls to go resident emanate from.

(e) File Viruses
This group contains viruses using the Operating System file system in one way or the other for replication purposes. The infection method of a file virus varies from one type to the other. They infect COM, EXE, data, SYS, OBJ, LIB and even text files. These are classified as Overwriting, Parasitic, Companion, Link, VBS, OBJ and LIB Viruses. This class of virus tends to attack a set of system files that are known and their detection by their objective may not be difficult (Patrick, 1992; Gordon, 1994 and Hanau, 1997). We assume that, it is illegal to change the content of all system files except when the operating system is being upgraded. Therefore, an attempt to change SYS, EXE, COM, OBJ, LIB, files etc. when not initiated by corresponding owners, i.e. the O/S, Compiler, etc. will be interpreted as illegal and adequate provision made for its blockage.

Algorithm 1.

```
Virus(void)                /* Virus main body */
{
        infectExecutableFile();
        if ( triggered( ) {
                doDamage( );
        }
        goto main of infected program/file
}

Void infectExecutable( void ) /* Searches for unifected executable file */
{
        file = choose an unifected executable file;
        prepend  Virus to file
}
void damage ( void ) /* The payload */
{
  do anything
}
int triggered ( void )  /* Triggers virus load */
{
        return ( some test ? 1 : 0 ) ;
}
```

Fig. 1: The structure of a file infecting virus

4. Virus Anti-Detection Techniques

In order to achieve their targets, viruses often employ special techniques to shield themselves from detection. Also, because the same anti-virus software techniques are in use, virus writers have attempted to defeat this software in their viruses by one or both of these techniques (Patrick, 1992 and Paget, 2000):

i Disabling the software or any of its modules.

ii Getting around the detection algorithms.

iii Removing the antivirus signature database.

Consequently, this has resulted into virus classification based on their anti-detection technique. We briefly discuss below some of these techniques.

5. Polymorphic Viruses

The scanning techniques used in most of the anti-virus software rely heavily on virus signature recognition. Thus virus writers have resulted into making their viruses capable of changing their codes after each infection session. This will eventually make the virus untraceable, as it would be difficult to get its correct signature into the database of the scanner (Paget, 2000 and Leitold, 2001).

(a) Tunneling
This is another technique often used to ensure that the virus is loaded underneath the scanner. This class of viruses often target interrupt handlers and thus have direct access to the operating system (Patrick, 1992).

(b) Other Techniques
Some of the viruses being developed today use a combination of the above techniques and a few more of their own. Some viruses would upon launch, disable anti-virus software or corrupt the "*.dbf" files containing virus signatures used by the scanners. Others will reduce their searching techniques with improved algorithms.

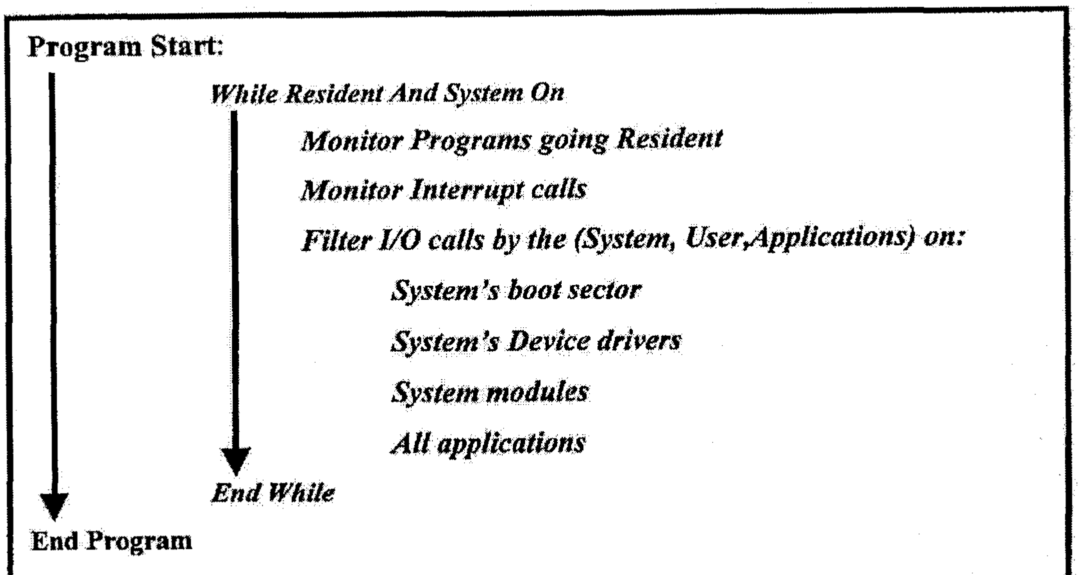
Generally, a behavioural detector would be able to catch these viruses as most of these virus activities could be used to distinguish viruses from non-viruses (Patrick, 1992 and Lu, 2002).

6. Descriptive Model of the Detector

We present a brief overview of the 'Behavioural Detector' and the tools to be used. The program is designed to be compact enough to be able to reside in memory while retaining all the proposed functionality. The most ideal language for its implementation is the C or Assembly programming language because of its closeness to the machine language and the possibility of combining both languages. Generally, an object-oriented language is not ideal for this type of work because of the size of its executable codes and their characteristic machine dependency structure. A general architectural model for the detector is presented in (Fig. 2).

7. Software and Hardware Interrupts

The concept of interrupts has expanded in scope over the years. The MSDOS and Windows family of Operating System have incorporated a lot of confusion into this phenomenon. Virus writers have explicitly explored the absence of rigidity to penetrate the Operating System and wreck havoc. In the 80X86 families, there are three types of events commonly known as interrupts and these are *exceptions, traps and hardware interrupts.* Although (the terms) *trap* and *exception* are often used interchangeably, in the context of this paper, the latter refers to a programme initiated and expected transfer of control to a special handler routine and its nothing more than a specialized subroutine call (Leitold, 2001). The 80X86-instruction *int* call is the vehicle for executing a trap. It is important to state here that traps are usually unconditional and when an *int call* is executed, control is transferred to the procedure associated with the trap. Therefore, since traps are executed via an explicit instruction, it is easy to determine exactly which instructions in a program will invoke a trap *handler* routine. On the other hand, an *exception* is an automatically generated trap and occurs in response to some exceptional condition generated due to exceptional behaviour of normal 80X86 instruction. Whenever this occurs, the CPU normally suspends the currently executed instruction and transfer control to the *exception handler* routine.



```
Program Start:
        While Resident And System On
                Monitor Programs going Resident
                Monitor Interrupt calls
                Filter I/O calls by the (System, User,Applications) on:
                        System's boot sector
                        System's Device drivers
                        System modules
                        All applications
        End While
End Program
```

Fig: 2: A General Architectural Model Of The Detector

Although there is no specific instruction to cause an exception, as with the software interrupts (Int traps), execution of some instructions always causes exceptions. As an example, executing a division instruction causes a division error or request for inaccessible memory region. Hardware interrupts in this work handles some events such as; Keyboard, Timer, Disk etc. and inform the CPU that a device needs some attention. The CPU interrupts the currently executing program, services the device, and then returns control back to the program. An interrupt service routine is a specifically written procedure provided by the Operating System to handle a *trap, exception and interrupt.* Although different phenomena cause traps, exceptions and interrupts, an ***Interrupt Service Routine or ISR*** has the same structure for each of these three (Jones, 1998). Generally, the structure of the ISR in the DOS environment has provided virus writers a readily available entry point for the activation of viruses and allowing these malicious programs to replicate with ease. An effective technique of putting these ISR's under control and close monitoring will surely ensure the curtailment of these viruses and their activities in the DOS environment. We need to block the possibilities of allowing access to target ISR's by these virus writers.

Trapping the needed Interrupts

Perhaps the most interesting aspect of the generic behavioural detector is the technique of blocking access to the Interrupt routines. Although this technique is not peculiar to this work, virus writers often employ the same technique. For all the needed interrupts, we save the addresses of their associated routines (systems) and insert the address of the detector's handling routine. Inside the detector's handling routine, we then try to determine the genuineness or otherwise of the calling routine's request (Fig. 3). Hardware interrupts are handled by Interrupt Service Routines (ISR) that correspondingly instruct the processor to carry out the task requested by the hardware such as Floppy Drives, Hard Drives, Timer, etc. and are potential entry points to triggering a malicious program. In addition to the known hardware interrupts i.e. generated by Hard and Floppy Drives, Timer, etc., that is, those that could assist the replication requirements of viruses, we also need to monitor such BIOS (Basic Input and Output System) requests via the INT 21H calls (Patrick, 1992).

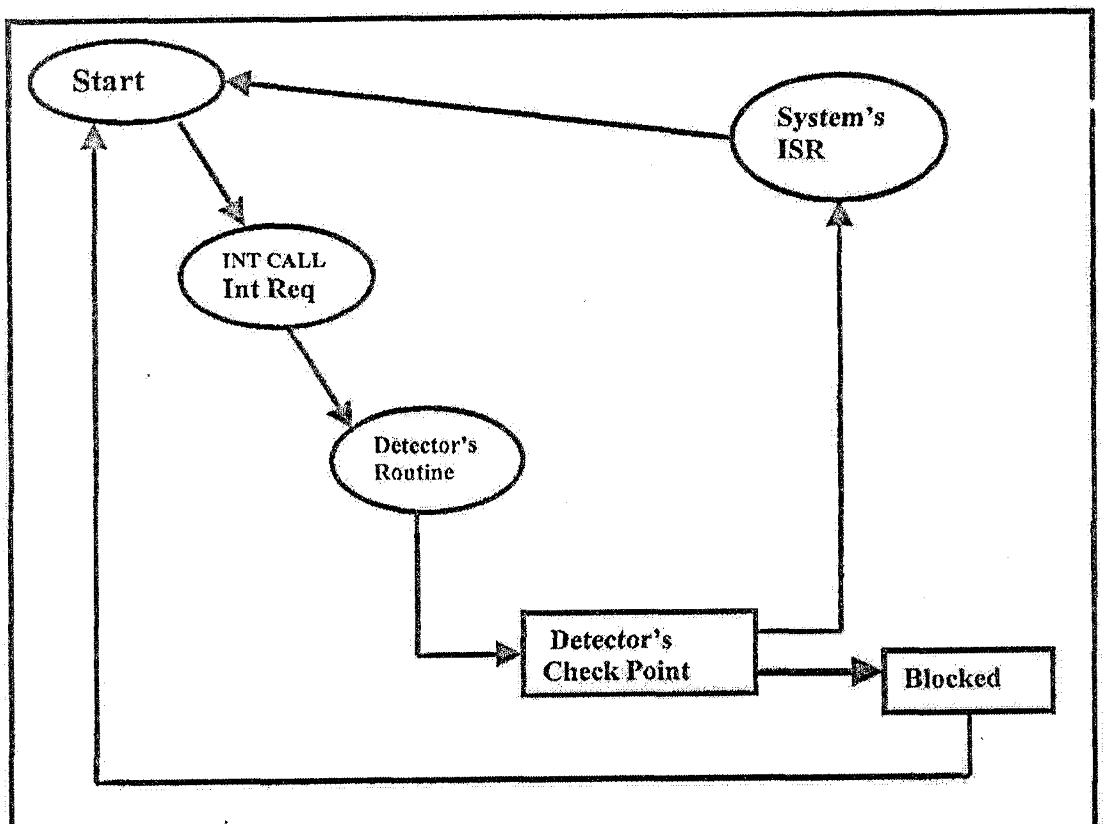


Fig 3: Schematic Representation Of GVBD

8. Necessary Components of the GVBD

The three major components of the GVBD are:

- i Identifying the system calls that are susceptible to virus attacks.
- ii The GVBD presentation as a TSR (Terminate and Stay Resident)
- iii Getting the GVBD into memory immediately after the Operating System is loaded.

(a) The BIOS and DOS Calls
Our first approach is to decide on the various calls that are potential entry points to virus attacks and to be monitored against malicious use. Some of the interrupt vectors, BIOS and DOS calls that could be used as launch pads and channels by viruses are:

INT 13H – BIOS disk services
INT 10H – BIOS video services
INT 21H – DOS services
INT 1AH – BIOS Real Time clock services
INT 8 (IRQ 0) Timer Interrupt
INT 9 (IRQ 1) Keyboard interrupts
INT 0Eh (IRQ 6) Diskette drive.Interrupt
INT 76h (IRQ 14) Hard Disk drive Interrupt

(b) The GVBD as a Terminate and Stay Resident Program
The GVBD program would be resident in memory as a "*terminate and stay resident*" program (TSR) and this must be realised just after loading the Operating System and its components in order to capture all the system resources before any virus action. This will allow an effective monitoring of all the activities of the computer system.

(c) Further Work Enhancement
Studies have proved a virus behavioural detector to be more effective than a virus scanner because of the need to continually update the scanner's database for newly discovered signatures. However, for monitoring to be effective users must be educated to be able to identify normal system activities from suspected activities. Also, the sensitivity of the monitor has to be set high and appropriate so as to reduce the generation of false alarms.

9. Conclusion

The proposed GVBD is an effective tool against all classes of viruses and even those that could only be detected after attacking at least one system. While the conventional anti-virus scanner relies heavily on virus signature or strains for detection, the GVBD looks for any form of abnormal behaviour of a program and therefore has the potential of being more effective and relevant in the fight against malicious codes. The idea however is not to abandon signature-based anti-virus software; but instead, create a more robust malicious code protection solution with both techniques; signature-based and behavioural-based (GVBD).

Acknowledgement

The authors thank Professor Amos David of LORIA, Toulouse, France for his commitment towards the upliftment of Information Communication Technology (ICT) in Nigerian Universities and his encouragement towards the realisation of this paper.

REFERENCES

Conry-Murray, A., 2002. Behavior-Blocking Stops Malicious Code, *Network Magazine*, June 2002 CMP Media, N.Y.

Gordon, S., 1994. The Generic Virus Writer. In: *Proceedings of the Fourth International Virus Bulletin Conference*, Jersey, U.K.

Gordon, S., 1995. Technologically Enabled Crime: Shifting Paradigms for the year 2000. *Computers and Security*, Elsevier Press, December 1995.

Hanau, 2000. Building an Anti-Virus Engine. http://www.hanau.net/fgk/downloads/42.zip.

Jones, D., 1998. 80X86 Assembly Programming. Oxford University Press, Oxford.

Leitold, F., 2001. Reductions of the General Virus Detection Problem, *EICAR Conference 2001*. http://papers.weburb.net/archive/00000063.

Lu, L., 2002. A Simplified Approach For Anti-Virus Implementation on EMAIL Servers. *EICAR Conference 2002*. http://papers.weburb.net/archive/00000072.

Munro, J., 2002. Anti-Virus Research and Detection Techniques http://www.extremetech.com/article2/0,3973,325897,00.asp.

Paget, F., 2000. Computer Viruses: The Technological Leap, *EICAR Conference*, 2000

Patrick, M., 1992. Virus Detection Alternatives. Published by the Dutch National Criminal Intelligence Service, Computer Crime Unit, The Hague, Netherlands.

Philip, F. and Thomas, D., 1998. The Computer Virus Crisis, Van Nostrand Reinhold, New York.

Vprotect, 2002. Computer Knowledge Virus Tutorial, http://www.cknow.com/vtutor/vprotect.htm, Computer Knowledge.com.